

7. Meanings for complex structures

7.1 Introduction

In Chapter 6, we introduced the notion a new metalanguage, **TY**. In this Chapter, we'll flesh it out.

This will allow us to propose meanings for *any* kind of constituent, including VPs, NPs, adverbs, adjectives, prepositions, connectives and so on.

We can also replace our strategy of invoking 'construction specific rules' (e.g., like Rule Q and Rule D, and rules for each connective), and unify everything under one kind of system.¹

For this reason, the material in this chapter will serve to link everything together so we can build a fully specified semantic grammar for some fragment of English.

7.2 Functions

7.2.1 Basics of functions

As in the previous handout, the basis of this theory is the notion of 'function'.

A function is a special type of relation, i.e., a set of ordered pairs. Specifically, a function is a relation R , such that if $\langle x, y \rangle \in R$ and $\langle x, z \rangle \in R$, then $y = z$. That is, no element can be paired with more than one element.

When talking about a function, and $\langle x, y \rangle \in R$, I'll say the following equivalent things:

- (1) a. "x maps to y (in R)"
- b. "R maps x to y"
- c. most importantly: " $R(x) = y$ "

Here's a specification of a function. The 'domain' is the set of things on the left, and the 'range' is the set of things on the right.

¹This goal is called 'type-driven translation' (Sag and Klein 1981), we will see why.

$$(2) \quad R = \left[\begin{array}{l} \text{😊} \mapsto \mathbf{T} \\ \text{😬} \mapsto \mathbf{T} \\ \text{😄} \mapsto \mathbf{F} \\ \text{😇} \mapsto \mathbf{T} \end{array} \right]$$

We could also say that R is a function “from” $\{\text{😊}, \text{😬}, \text{😄}, \text{😇}\}$ “to” $\{\mathbf{T}, \mathbf{F}\}$.

7.2.2 Types of functions

- (3) A relation R is a *function* iff each x in the domain of R is mapped by R to at most one element in the range of R . (a function cannot ‘doubly map’ anything).

Describe a relation which is *not* a function.

- (4) A function f is *total* iff every element in the domain of f has a value in the range of f . If f fails to meet this condition, it is called a *partial* function.

Is (2) a total or partial function? What about this one?

$$(5) \quad R = \left[\begin{array}{l} \text{😊} \mapsto \mathbf{T} \\ \text{😬} \mapsto \mathbf{T} \\ \text{😄} \mapsto \mathbf{T} \\ \text{😇} \mapsto \mathbf{T} \end{array} \right]$$

- (6) A function f is *onto* iff every element in the range of f is the value of some element in the domain of f .

Is (5) onto?

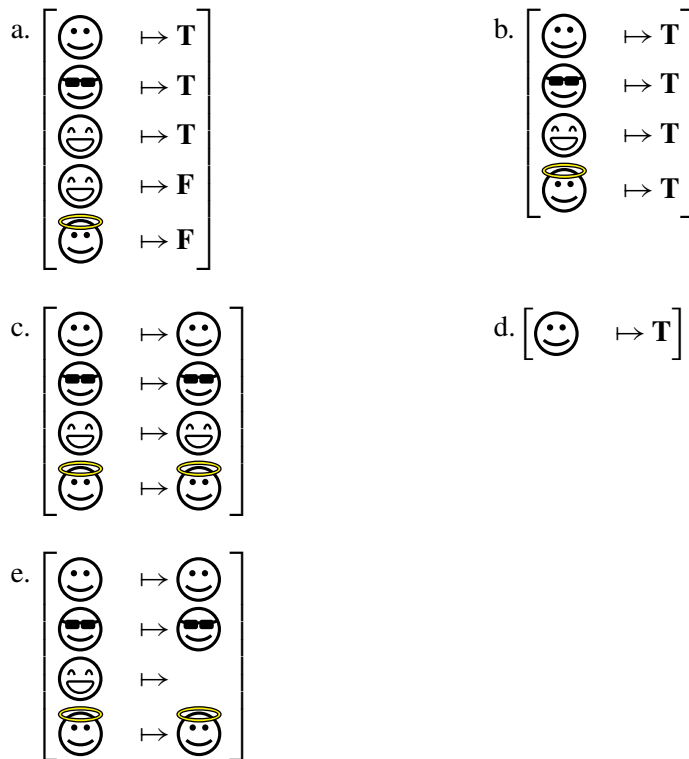
- (7) A function f is *one-to-one* iff no member of the range is assigned to more than one member of the domain.

Is the function below *one-to-one*?

$$(8) \quad R = \left[\begin{array}{l} \text{😊} \mapsto \text{😇} \\ \text{😬} \mapsto \text{😄} \\ \text{😄} \mapsto \\ \text{😇} \mapsto \text{😬} \end{array} \right]$$

- (9) A function f is *bijective* or a one-to-one correspondence iff it is total, onto, and one-to-one. (i.e., there’s a unique mapping for each member of the domain, and for each member of the range.)

- (10) For each of the following, say whether it’s a function, and if yes whether it is total and/or onto (adapted from Potts 2007). Assume the domain is $\{\text{😊}, \text{😬}, \text{😄}, \text{😇}\}$ in each case.



- f. the relation R from nodes to nodes in tree structures that maps each node to its daughter(s)
 g. the relation R^1 from nodes to nodes in tree structures that maps each node to its mother(s)
 h. a function from x^2 to the value(s) of x
 i. a function from x to the value(s) of x^2

We can specify the domain and range of a function using the following notation: $f : A \mapsto B$, i.e., the function f maps members of A to members of B .

Which of the functions in (10) are specified $f : U \mapsto \{\mathbf{T}, \mathbf{F}\}$? How would you specify the rest?

A useful concept: *function composition*.

(11) The composition of $f : A \mapsto B$ and $g : B \mapsto C$, is a function $g \circ f : A \mapsto C$.

$$g \circ f(d) = g(f(d))$$














What would be the value of $b \circ c(\text{☹})$, or equivalently $b(c(\text{☹}))$.

(12) (from Benthem et al. 2016)

The successor function is $s : \mathbb{N} \mapsto \mathbb{N}$ is given as $s(n) = n + 1$. How would you characterize the composition of s with itself?

7.3 Types

The function-based approach to semantics relies on a notion of ‘types’. Implicitly, we have categorized expressions in our metalanguage by the kinds of objects they denote in the model.

	Expression	Category	ML Translation	Denotation
	<i>it's raining</i>	S	rain!	T
	<i>Smiley</i>	DP	smiley	
(13)	<i>skateboards</i>	V _{intr}	skateboards	{  ,  }
	<i>teases</i>	V _{tr}	teases	{ {  ,  }, {  ,  } }
	<i>introduces</i>	V _{ditr}	introduces	{ {  ,  ,  }, {  ,  ,  } }
	<i>every</i>	D	every	{ { P, Q } $P \subseteq Q$ }

This is essentially what is meant by typing: this way of classifying the expressions in the metalanguage. So let's see how to do this in depth, *and* extend it to any kind of expression.

7.3.1 The basic types

Our metalanguage is named **TY** (from Gallin 1975), or 'type theory'. Here, we start off with two basic types for expressions.

We have introduced the two basic types already. The first basic type is t . t is assigned to any ML expression which denotes a truth value (t for truth value). Which of the following expressions are type t ?

- (14)
- rain!**
 - teases(smiley)**
 - smiley**
 - $\forall x[\text{annoyed}(x) \rightarrow \exists y[\text{proud-of}(x, y)]]$
 - the(saxophonist)**
 - \wedge
 - angel**
 - $\exists x[\text{skateboards}x] \rightarrow \forall y[\text{annoyed}(y)]$

The other basic type is e (for entity). e is assigned to any ML expression which denotes an individual. Which of the above expressions are type e ?

These are the basic types because the denotations cannot be deconstructed into more basic types.

- (15)
- $\llbracket \text{smiley} \rrbracket = \text{☺}$
 - $\llbracket \text{the(saxophonist)} \rrbracket = \text{🎷}$ type e
 - $\llbracket \text{frowny} \rrbracket = \text{☹}$
- (16)
- $\llbracket \text{rain!} \rrbracket = \mathbf{F}$
 - $\llbracket \text{annoyed(smiley)} \rrbracket = \mathbf{F}$ type t
 - $\llbracket \exists x[\text{skateboards}(x)] \rrbracket = \mathbf{T}$

In order to keep things uniform: for the set of possible truth values, $\{\mathbf{T}, \mathbf{F}\}$, we can write D_t (the domain of truth values). For the set of possible individuals U , we can write D_e (the domain of entities).

The following are equivalent, $\{\mathbf{T}, \mathbf{F}\} = D_t$ and $U = D_e$.

7.3.2 Characteristic functions

In the last handout, we motivated the need for a notion of functions in our model.

It turns out we can replace the sets of individuals (used for denotations of predicates like **skateboards**) with isomorphic ‘characteristic functions.’ So let’s figure out how to do that.

The characteristic function of set A from some domain U is a function that maps all members of A to **T** and all elements of U that are not members of A to **F** (from Benthem et al. 2016).

For example, the function representing the property of being divisible by 3, on the domain of integers, would map the numbers $\dots, -9, -6, -3, 0, 3, 6, 9, \dots$ to **T**, and all other integers to **F**.

By definition, we can find a unique characteristic function (or CF) for each set of individuals. The set of CFs $f : D_e \mapsto D_t$ is *isomorphic* to $\wp(U)$.

$$(17) \quad \left[\begin{array}{l} \text{😊} \mapsto \mathbf{T} \\ \text{😬} \mapsto \mathbf{T} \\ \text{😞} \mapsto \mathbf{F} \\ \text{😊} \mapsto \mathbf{T} \\ \text{😞} \mapsto \mathbf{F} \\ \text{😞} \mapsto \mathbf{T} \end{array} \right] \iff \{ \text{😊}, \text{😬}, \text{😊}, \text{😞} \}$$

$$(18) \quad \left[\begin{array}{l} \text{😊} \mapsto \mathbf{F} \\ \text{😬} \mapsto \mathbf{T} \\ \text{😞} \mapsto \mathbf{T} \\ \text{😊} \mapsto \mathbf{F} \\ \text{😞} \mapsto \mathbf{T} \\ \text{😞} \mapsto \mathbf{T} \end{array} \right] \iff$$

$$(19) \quad \iff \{ \text{😊}, \text{😞}, \text{😞} \}$$

Another specifying the domain and range of a function is using superscript, such that $A \mapsto B$ can be written as B^A . This notation makes sense when start to think about the numbers of functions involved (from Potts 2007).

- (20) Let $A = \{ \text{😊}, \text{😊}, \text{😬} \}$ and $B = \{ \mathbf{T}, \mathbf{F} \}$.
- How many total functions f are there such that $f : A \mapsto B$?

- b. How many objects are there in $\wp(A)$? This result helps us understand why the powerset is sometimes written as 2^A .
- c. How many objects are in $A \times B$?

A rough and ready definition for CFs:

(21) **Characteristic function:**

A function f from a set D_e to the set D_t is a characteristic function iff for every $d \in U$:

$$f(d) = \begin{cases} \mathbf{T} & \text{if } d \in U \\ \mathbf{F} & \text{if } d \notin U \end{cases}$$

7.3.3 Characteristic functions as meanings of predicates

With no loss of information, we can switch up our denotations so that 1-place predicates map to CFs instead of sets. Here's a new specification of $\llbracket \cdot \rrbracket^M$.

$$(22) \quad \text{a. } \llbracket \text{smiles} \rrbracket^M = \left[\begin{array}{l} \text{😊} \mapsto \mathbf{T} \\ \text{😬} \mapsto \mathbf{F} \\ \text{😏} \mapsto \mathbf{F} \\ \text{😇} \mapsto \mathbf{F} \\ \text{😞} \mapsto \mathbf{T} \\ \text{😐} \mapsto \mathbf{T} \end{array} \right] \quad \text{b. } \llbracket \text{annoyed} \rrbracket^M = \left[\begin{array}{l} \text{😊} \mapsto \mathbf{F} \\ \text{😬} \mapsto \mathbf{T} \\ \text{😏} \mapsto \mathbf{F} \\ \text{😇} \mapsto \mathbf{T} \\ \text{😞} \mapsto \mathbf{F} \\ \text{😐} \mapsto \mathbf{T} \end{array} \right]$$

Because sets and characteristic functions (CFs) are isomorphic, we should be able to comfortably switch between them as the need arises.

Formerly, this was our rule for interpreting sentences of **FOL** consisting of a 1-place predicate (like **annoyed**) and an individual argument (like **frowny**).

- $$(23) \quad \text{a. } \llbracket P(a) \rrbracket = \mathbf{T} \text{ iff } \llbracket a \rrbracket^M \in \llbracket P \rrbracket^M$$
- $$\text{b. } \llbracket \text{annoyed}(\text{frowny}) \rrbracket = \mathbf{T} \text{ iff } \llbracket \text{frowny} \rrbracket^M \in \llbracket \text{annoyed} \rrbracket^M$$

But now, **annoyed** denotes a CF. There's a direct connection between the ML and the denotation.

- $$(24) \quad \text{a. } \llbracket P(a) \rrbracket^M = \mathbf{T} \text{ iff } \llbracket P \rrbracket^M(\llbracket a \rrbracket^M) = \mathbf{T}$$
- $$\text{b. } \llbracket \text{annoyed}(\text{smiley}) \rrbracket^M = \mathbf{T} \text{ iff } \llbracket \text{annoyed} \rrbracket^M(\llbracket \text{smiley} \rrbracket^M) = \mathbf{T}$$


What are the values of the following?

- $$(25) \quad \text{a. } \llbracket \text{smiles} \rrbracket^M(\llbracket \text{toothy} \rrbracket^M) =$$
- $$\text{b. } \llbracket \text{annoyed} \rrbracket^M(\llbracket \text{smiley} \rrbracket^M) =$$
- $$\text{c. } \llbracket \text{annoyed} \rrbracket^M(\llbracket \text{angel} \rrbracket^M) =$$

So 1-place predicates denote CFs, functions which map individuals in D_e to truth values in D_t .

The set of all functions from D_e to D_t (i.e., the set of all CFs) is written as $D_{\langle e,t \rangle}$.

Because 1-place predicates like **skateboards** and **smiles** denote objects in $D_{\langle e,t \rangle}$, we assign them the type $\langle e,t \rangle$. This table summarizes what we have so far.

	A TY expression like...	...is type...	...so its denotation is in ...	An example denotation
	rain!	t	D_t	T or F
	smiley	e	D_e	
(26)	smiles	$\langle e, t \rangle$	$D_{\langle e, t \rangle}$	$\left[\begin{array}{l} \text{smiley face} \mapsto \mathbf{T} \\ \text{frowny face} \mapsto \mathbf{F} \\ \dots \end{array} \right]$

7.3.4 Defining possible types

Repeating what's written above: the set of all functions from D_e to D_t is written as $D_{\langle e, t \rangle}$.

Here we took two sets of objects A and B , and defined a third one as the set of functions mapping from objects in A to objects in B . We can keep doing this to define more and more types of expressions.

(27) **Possible Types:**

- e is a type
- t is a type
- If σ and τ are types, then $\langle \sigma, \tau \rangle$ is a type.
- Nothing else is a type.

Some examples of types for metalanguage expressions

- (28)
- e, t
 - $\langle e, e \rangle, \langle t, t \rangle, \langle e, t \rangle, \langle t, e \rangle$
 - $\langle e, \langle e, e \rangle \rangle, \langle e, \langle t, t \rangle \rangle, \langle e, \langle e, t \rangle \rangle, \langle e, \langle t, e \rangle \rangle, \langle t, \langle e, e \rangle \rangle, \langle t, \langle t, t \rangle \rangle, \langle t, \langle e, t \rangle \rangle, \langle t, \langle t, e \rangle \rangle$
 - $\langle \langle e, e \rangle, e \rangle, \langle \langle t, t \rangle, e \rangle, \langle \langle e, t \rangle, e \rangle, \langle \langle t, e \rangle, e \rangle, \langle \langle e, e \rangle, t \rangle, \langle \langle t, t \rangle, t \rangle, \langle \langle e, t \rangle, t \rangle, \langle \langle t, e \rangle, t \rangle$
 - $\langle \langle e, e \rangle, \langle e, e \rangle \rangle, \langle \langle e, e \rangle, \langle t, t \rangle \rangle, \langle \langle e, e \rangle, \langle e, t \rangle \rangle, \langle \langle e, e \rangle, \langle t, e \rangle \rangle, \langle \langle t, t \rangle, \langle e, e \rangle \rangle, \langle \langle t, t \rangle, \langle t, t \rangle \rangle, \langle \langle t, t \rangle, \langle e, t \rangle \rangle, \langle \langle t, t \rangle, \langle t, e \rangle \rangle$
 -and so on

So an expression may have any one of the above types or an infinite number of other types. Why are the following not possible types in **TY**?

- (29)
- $\langle e, e, t \rangle$
 - $\langle s, t \rangle$

This kind of definition of types in (27) is totally standard, such that semanticists often gloss over it, but it's worth exploring a bit to get a handle on it (from Potts 2007).

(30) Consider the following definition of a (toy) type space:

- \circ and \bullet are types
- If σ is a type, then $\langle \dagger, \sigma \rangle$ is a type.
- Nothing else is a type.


Are the following in this type space?

- (31)
- $\langle \dagger, \circ \rangle$
 - $\langle \circ, \dagger \rangle$
 - $\langle \circ, \bullet \rangle$
 - $\langle \dagger, \langle \dagger, \langle \dagger, \circ \rangle \rangle \rangle$

The type definition in (27) will be used to categorize every expression of **TY**. No expression can have a type not derived by (27).

7.3.5 Denotations for typed expressions

How do we know what expression has what type? By the kind of object it denotes.

The expression **smiley** is type e , because its denotation () is a member of D_e . The expression **rain!** is type t because its denotation (**T**) is a member of D_t .

We always know what type an expression is by its denotation. If the denotation of an expression is a member of D_X , then the expression is type X .

For this reason, models need to supply a set D_X for every type X . These sets are called ‘domains’, not to be confused with the domain of a function. I recognize this is confusing, so I’ll spell these as ‘d-domains’ (for denotation-domain perhaps).

- (32) **Possible d-domains:**
- D_e is a d-domain (i.e., U)
 - D_t is a d-domain (i.e., $\{\mathbf{T}, \mathbf{F}\}$)
 - If D_σ and D_τ are d-domain, then $D_{\langle \sigma, \tau \rangle}$ is a d-domain.
 - Nothing else is a d-domain.

Now there is a set of denotations for expressions of every type defined in (27).

- (33) What are all the functions in the d-domain $D_{\langle t, t \rangle}$?

What type is \neg ? What is $\llbracket \neg \rrbracket$?

- (34) Let’s say $D_e = \{ \text{smiley}, \text{frowny} \}$. There is a set of function $D_{\langle \langle e, e \rangle, t \rangle}$. What is the domain of these functions (i.e., the set of possible inputs)? How many functions are in $D_{\langle \langle e, e \rangle, t \rangle}$?

- (35) Let’s say $D_e = \{ \text{smiley}, \text{frowny} \}$. Give an example of a function in $D_{\langle \langle e, t \rangle, t \rangle}$.

7.4 Function application

Function by nature take inputs and spit out outputs. It's with this simple intuition that we model how the meanings of complex constituents in NL are composed from the meanings of smaller sub-constituents.

For example, the meaning of the sentence **smiles(smiley)** is type t . It has two sub-constituents: **smiles** of type $\langle e, t \rangle$ and **smiley** of type e .

TY allows us to combine these two sub-constituents to make the larger one by *function application*. We'll use the symbol $+$ for functional application, following Winter 2016.

$$(36) \quad \mathbf{smiles}_{\langle e, t \rangle} + \mathbf{smiley}_e = \mathbf{smiles(smiley)}_t$$

Function application ($+$) is commutative, so:

$$(37) \quad \mathbf{smiley}_e + \mathbf{smiles}_{\langle e, t \rangle} = \mathbf{smiles(smiley)}_t$$

A generalized definition for function application:²

$$(38) \quad \mathbf{Function\ application}$$

$$f_{\langle \sigma, \tau \rangle} + a_\sigma = a_\sigma + f_{\langle \sigma, \tau \rangle} = f(a)_\tau$$

For each of the following, say whether the $+$ operation is defined, and if so, what is the type?

- (39)
- $e + \langle e, \langle e, t \rangle \rangle$
 - $\langle \langle e, \langle e, t \rangle \rangle, \langle e, t \rangle \rangle + \langle e, e \rangle$
 - $\langle \langle e, \langle \langle e, t \rangle, t \rangle \rangle, \langle t, t \rangle \rangle + e$
 - $\langle \langle e, t \rangle, t \rangle + \langle e, t \rangle$
 - $\langle e, \langle e, t \rangle \rangle + \langle \langle e, \langle e, t \rangle \rangle, \langle e, t \rangle \rangle$
 - $\langle \langle e, e \rangle, \langle e, t \rangle \rangle + e$

What is the value (plus its type) of each of the following?

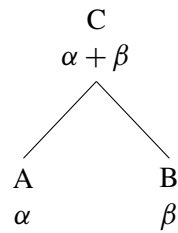
- (40)
- $\mathbf{the-mother-of}_{\langle e, e \rangle} + \mathbf{cool}_e =$
 - $\mathbf{frowny}_e + \mathbf{teases}_{\langle e, \langle e, t \rangle \rangle} =$
 - $\mathbf{snowman}_{\langle e, t \rangle} + \mathbf{talented}_{\langle \langle e, t \rangle, \langle e, t \rangle \rangle} =$
 - $\mathbf{teases}_{\langle e, \langle e, t \rangle \rangle} + \mathbf{snowman}_{\langle e, t \rangle} =$

When $A + B$ is undefined, we call it a 'type mismatch'. Type mismatches are an extremely useful way to account for ungrammaticality and blocked readings (for example, see Collins 2018).

$+$ gives us a very powerful tool for mapping syntax to semantics, according to this schema:

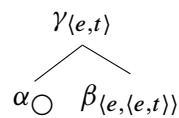
- (41) **Interpreting a branching tree:**
 Let C be a non-terminal node in a tree with daughters A and B .
 If $A \rightsquigarrow \alpha$ and $B \rightsquigarrow \beta$, then $C \rightsquigarrow \alpha + \beta$.

²The calculus defined by (38) is an Ajdukiewicz Calculus.



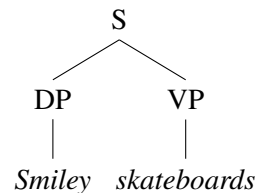
This theory of the syntax-semantics interface has a strict logic, and forces decisions on us for better or worse. This is important to get the hang of for problem solving/reasoning about the denotations.

(42) What is the type of α ? NB: there are two answers.



7.4.1 Interpreting intransitives

Now we have enough to build a theory of intransitive sentences. Take a tree:



We can figure out the ML translation, and the denotation of every node in the tree.

(43) **Terminals:**

If a node dominates just a lexical item and nothing else, its translation is the translation of the lexical item.

- (44) a. $DP \rightsquigarrow \mathbf{smiley}_e$
 b. $VP \rightsquigarrow \mathbf{skateboards}_{(e,t)}$

The translation of non-terminals is simply calculated via function application.

(45) $S \rightsquigarrow \mathbf{skateboards}(\mathbf{smiley})_t$

In terms of denotation, the interpretations of simple expressions are supplied by $\llbracket \cdot \rrbracket^{M,g}$.

- (46) a. $\llbracket \mathbf{smiley} \rrbracket^{M,g} = \text{☺}$
 b. $\llbracket \mathbf{skateboards} \rrbracket^{M,g} = \left[\begin{array}{l} \text{☺} \mapsto \mathbf{T} \\ \text{☹} \mapsto \mathbf{F} \end{array} \right]$

For complex expressions, we appeal the following rule.

(47) **Interpreting functions w/ arguments:**

$$\llbracket \alpha(\beta) \rrbracket = \llbracket \alpha \rrbracket(\llbracket \beta \rrbracket)$$

(48) $\llbracket \text{skateboards}(\text{smiley}) \rrbracket^{M,g} =$

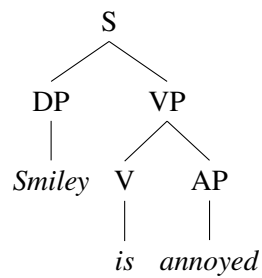
What about adjectival predicates? Let's define a meaning for English *be* in its function selecting adjectival and nominal predicates.

(49) $is \rightsquigarrow \mathbf{id}_{\langle \langle e,t \rangle, \langle e,t \rangle \rangle}$

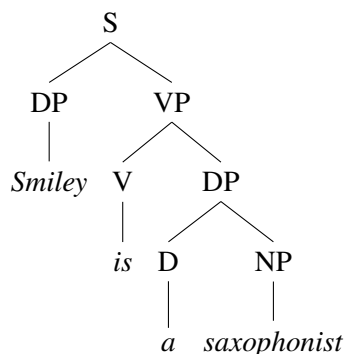
id denotes a function which maps any CF to itself. What would this look like visually?

- (50) a. **id(annoyed) = annoyed**
 b. **id(cat) = cat**

What are the translations (plus types) of the nodes in this tree?



What about nominal predicates? What are the translations and types of the nodes in this tree? You'll have to propose a semantics for *a*.

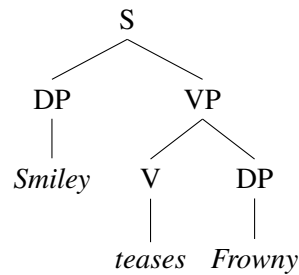


7.4.2 Transitive predicates

So far so easy. Transitive predicates are a little trickier as they have two arguments. Therefore, when they combine with their first argument, they are not yet truth value denoting.

We take ML expressions like **teases** to be type $\langle e, \langle e, t \rangle \rangle$. They need two arguments to be type t .

What is the type and value of **teases + smiley**? What about **(teases + smiley) + frowny**? What is the ML translation of each node in the following tree?



The translation of S is **(teases(frowny))(smiley)**. We usually omit the first set of brackets, so **teases(frowny)(smiley)**. Bare in mind this is “VOS” ordering (to borrow terminology from syntax).

The denotation of **teases** is slightly different to what we were working with in previous weeks.

$$(51) \quad (\text{previously:}) \llbracket \mathbf{teases} \rrbracket = \{ \langle \text{😐}, \text{😞} \rangle, \langle \text{😄}, \text{😎} \rangle, \langle \text{😞}, \text{😄} \rangle, \langle \text{😎}, \text{😐} \rangle \}$$

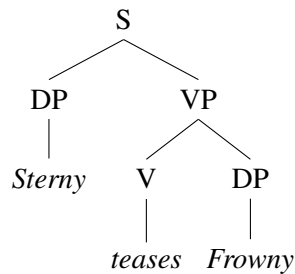
However, we can construct functions which are totally isomorphic to sets of ordered pairs. First, we can construct a function, isomorphic to (51), which maps each individual to the set of individuals who tease them. NB: this isn't a possible denotation in **TY**, but it's an intermediary step.

$$(52) \quad \llbracket \mathbf{teases} \rrbracket = \left[\begin{array}{l} \text{😞} \mapsto \{ \text{😐}, \text{😎} \} \\ \text{😐} \mapsto \emptyset \\ \text{😄} \mapsto \{ \text{😄} \} \\ \text{😎} \mapsto \{ \text{😐} \} \end{array} \right]$$

Now of course we know that each set of individuals has a corresponding CF. If we sub in the right CF above, we get a function from individuals to functions from individuals to truth values: $\langle e, \langle e, t \rangle \rangle$.

$$(53) \quad \llbracket \mathbf{teases} \rrbracket = \left[\begin{array}{l} \text{😞} \mapsto \left[\begin{array}{l} \text{😞} \mapsto \mathbf{F} \\ \text{😐} \mapsto \mathbf{T} \\ \text{😄} \mapsto \mathbf{T} \\ \text{😎} \mapsto \mathbf{F} \end{array} \right] \\ \text{😐} \mapsto \left[\begin{array}{l} \text{😞} \mapsto \mathbf{F} \\ \text{😐} \mapsto \mathbf{F} \\ \text{😄} \mapsto \mathbf{F} \end{array} \right] \\ \text{😄} \mapsto \left[\begin{array}{l} \text{😞} \mapsto \mathbf{F} \\ \text{😐} \mapsto \mathbf{F} \\ \text{😄} \mapsto \mathbf{T} \end{array} \right] \\ \text{😎} \mapsto \left[\begin{array}{l} \text{😞} \mapsto \mathbf{T} \\ \text{😐} \mapsto \mathbf{F} \\ \text{😄} \mapsto \mathbf{F} \end{array} \right] \end{array} \right]$$

With this function, let's evaluate the translations, types, denotations of each constituent below.



Previously we treated the denotations of transitives as sets of pairs, but now we use functions in $D_{\langle e, \langle e, t \rangle \rangle}$ like (53). In a compositional theory, these views make a difference. But, mathematically, these two types of denotations are indistinguishable.

This insight traces to M. Schönfinkel and is thus sometimes called Schönfinkelization. Start with a function $f : A \mapsto (B \mapsto C)$ and then define the equivalent function $f' : (A \times B) \mapsto C$ as follows:

$$f'(\langle a, b \rangle) = f(b)(a)$$

For linguists: $V'(\langle S, O \rangle) = V(O)(S)$

(54) (from Benthem et al. 2016):

- a. Describe a function $\geq : (\mathbb{N} \times \mathbb{N}) \mapsto D_t$.
- b. Describe the isomorphic function $\geq : \mathbb{N} \mapsto (\mathbb{N} \mapsto D_t)$

A linguistic theory might offer many different potential ways of explaining the deviance of some example. The semantic typing of expressions like verbs provides an insightful way to describe the unacceptability of a sentence.

(55) How would you explain the deviance in the following examples?

- a. *Ed devoured.
- b. *Ed glimpsed the dog the printer.

A linguistic theory could therefore provide multiple ways to explain unacceptability, some semantics, some syntactic, and so on. The difficulty is determining how to settle on one of the options, and it can be even more difficult to figure out how, or whether, to remove redundancies in one's account.

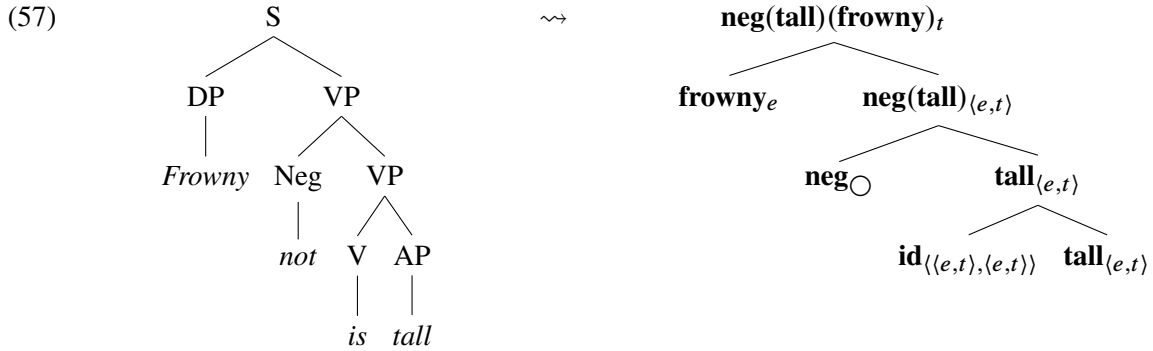
7.4.3 A case study: negation

We've already analyzed structures like the following.



Now we can revisit negation, in a way that is more sensitive to English syntax. Previously, our rule for negation was: where $S \rightsquigarrow \phi$, and S^- is the negated form of S , then $S^- \rightsquigarrow \neg\phi$.

It would be nice to get rid of this *non-compositional* S^- business. We can easily plot out the desired behavior of a VP-negation.³



(58) What type will this negative **neg**-operator be?

(59) $\llbracket \mathbf{neg} \rrbracket$ is a function which takes as its input a (a.) and returns as its output a (b.).

a. ??

b. ??

(60) For any function f of type $\langle e, t \rangle$, like $\llbracket \mathbf{tall} \rrbracket$, describe the function $\llbracket \mathbf{neg} \rrbracket(f)$.

(61) If $\llbracket \mathbf{tall} \rrbracket = \left[\begin{array}{l} \text{☺} \mapsto \mathbf{T} \\ \text{☹} \mapsto \mathbf{F} \\ \text{☺} \mapsto \mathbf{T} \end{array} \right]$, what is $\llbracket \mathbf{neg}(\mathbf{tall}) \rrbracket$?

Now we can just replace our S^- rule with the rule $not \rightsquigarrow \mathbf{neg}$.

7.5 Defining function-denoting expressions

Now we are comfortable using function-denoting expressions of various types. But so far we're limited to the ones supplied by the constants of our metalanguage, e.g., **teases**, **skateboards**, **neg**, **id** and so on.

But sometimes semanticists need to specify more complicated functions. Luckily **TY** gives us a way to do that, using the λ -operator.

7.5.1 Adding λ to the metalanguage

This is how we specified **id** above using English prose.

(62) $\llbracket \mathbf{id} \rrbracket$ is the function which maps a function f in $D_{\langle e, t \rangle}$ to itself.

The λ -operator allows us to write a ML expression which tells you exactly how to interpret it.

(63) a. $\lambda f_{\langle e, t \rangle}$ means “a function which maps a function f in $D_{\langle e, t \rangle}$...”

³For those of you worried about the word order, imagine we are analyzing Spanish/Italian/Mandarin.

- b. instead of “to itself” we can write $.f$
- c. therefore, we can rewrite **id** as $\lambda f_{\langle e,t \rangle}.f$

The symbol λ tells us that it is a function.

The dot separates the specification of the function’s argument and the definition of the function’s result. Before the dot, writing ‘ $f_{\langle e,t \rangle}$ ’ introduces ‘ f ’ as an ad hoc name for the argument of the function. The type $\langle e,t \rangle$ in the subscript of f tells us that this argument can be any object in the d-domain $D_{\langle e,t \rangle}$.

The re-occurrence of ‘ f ’ after the dot tells us that the function we defined simply returns the value of its argument. (paraphrased from Winter 2016)

Describe the following functions in words, and their types:

- (64)
- a. $\llbracket \lambda x_e . x \rrbracket$
 - b. $\llbracket \lambda P_{\langle e,t \rangle} . \mathbf{smiley} \rrbracket$
 - c. $\llbracket \lambda P_{\langle e,t \rangle} . P(\mathbf{smiley}) \rrbracket$

Let’s rewrite **neg** and **id** using lambdas.

- (65)
- a. **id** = $\lambda f_{\langle e,t \rangle} . f$
 - b. **neg** =

A more formal definition:

- (66) Let α be an expression of type τ , and u be a variable of type σ :
- a. then $\lambda u . \alpha$ is an expression of type $\langle \sigma, \tau \rangle$,
 - b. $\llbracket \lambda u . \alpha \rrbracket^g$ = the function $f : D_\sigma \mapsto D_\tau$, such that for all $d \in D_\sigma$, $f(d) = \llbracket \alpha \rrbracket^g[u \rightarrow d]$

7.5.2 Functional application with λ

To warm up, let’s just rewrite our trees with **neg** and **id**. Remember we interpret branching structures using $+$, so each time this will just involve putting one branch as the argument for the other.

- (67)
- ```

graph TD
 S --> DP
 S --> VP
 DP --> Frowny
 VP --> V
 VP --> AP
 V --> is
 AP --> tall

```

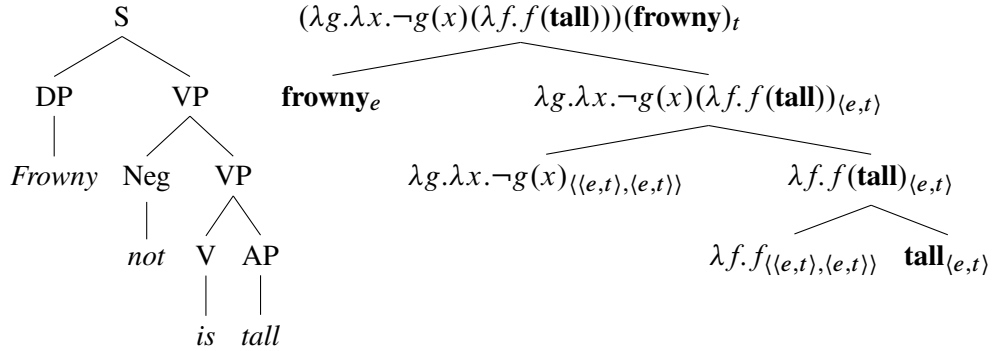
$\rightsquigarrow$

```

graph TD
 Root["lambda P.P(tall)(frowny)_t"] --> Frowny["frowny_e"]
 Root --> Node1["lambda P.P(tall)_e,t"]
 Node1 --> Node2["lambda P.P_e,t,e,t"]
 Node1 --> Tall["tall_e,t"]

```

(68)



Obviously these terms are almost unreadable. Luckily, we can use a way to abbreviate lambda terms called  $\beta$ -reduction.

(69)

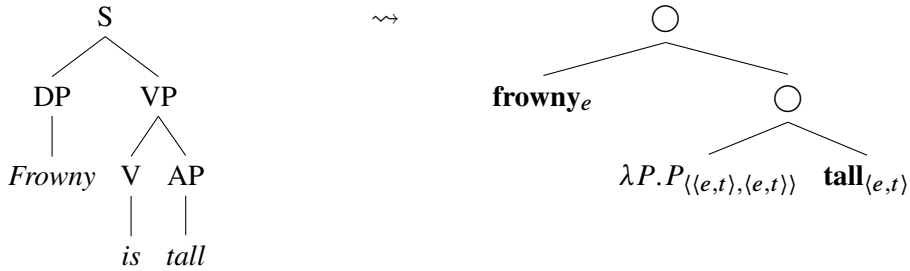
- $\beta$ -reduction** (informal):  
 Any term  $(\lambda u.\phi)(\psi)$  can be reduced by the following steps
- delete ' $\lambda u.$ '
  - delete ' $(\psi)$ '
  - replace any unbound instance of  $u$  inside  $\phi$  with  $\psi$ .

Some practice

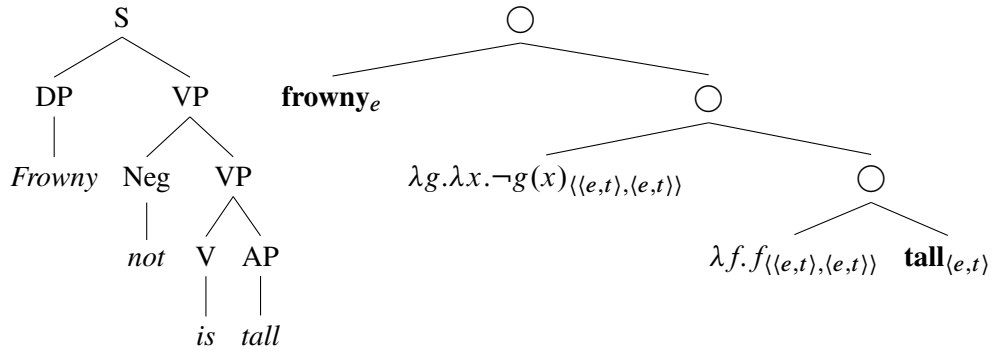
- $(\lambda x.\mathbf{annoyed}(x))(\mathbf{smiley})$
- $(\lambda x.\mathbf{like}(x))(y)$
- $(\lambda x.\mathbf{like}(y))(x)$
- $(\lambda x.\mathbf{run})(x)$
- $(\lambda P.\lambda x.P(\lambda y.\mathbf{admire}(y)(x)))(\lambda f.f(\mathbf{ali})(\mathbf{chris}))$

Armed with  $\beta$ -reduction, we can drastically simplify the trees above.

(71)

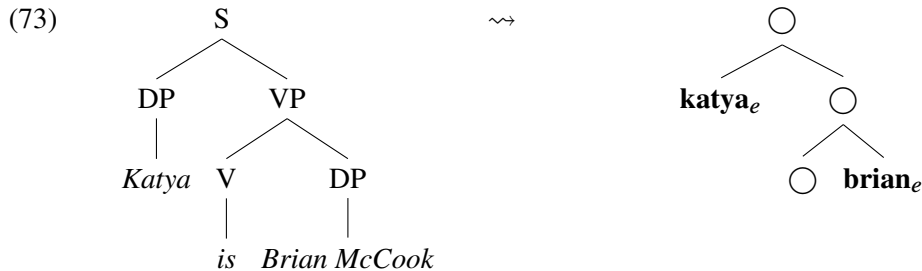


(72)

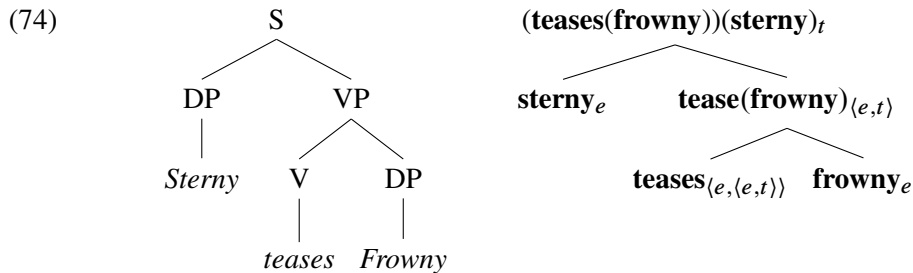




We also get English sentences with definite DP predicates. Propose a lexical entry for *is* here (assuming it's a different *is* from the  $\lambda f.f$  one above).



Question: do we need any  $\lambda$ s in our intransitive “Smiley danced” or transitive “Smiley teased Frowny” trees? Answer: No, in these cases, the constants are already functions, so no  $\lambda$ s are necessary.



Question: but I've seen semantics textbooks which give the meaning of *tease* as  $\lambda x.\lambda y.\mathbf{tease}(x)(y)$ , and the meaning of *cat* as  $\lambda x.\mathbf{cat}(x)$ . The answer is “eta-reduction”.

(75)  $\eta$ -reduction (informal): Any term  $\lambda u.\phi(u)$  is equivalent to  $\phi^4$

By  $\eta$ -reduction, the following are equivalent.

- (76)
- $\lambda x.\mathbf{dog}(x) = \mathbf{dog}$
  - $\lambda x.\lambda y.\mathbf{tease}(x)(y) = \mathbf{tease}$

“Lambdas are so closely associated with meaning analysis that it can be hard for people to see that they are not constitutive of a proposal about a particular denotation” (Potts 2007:86)

Intuitively, why is  $\eta$ -reduction guaranteed to work?

What's the difference between  $\lambda x.\mathbf{dog}(x)$  and  $\lambda y.\mathbf{dog}(y)$ ? What about  $\mathbf{dog}(y)$  and  $\mathbf{dog}(x)$ ?

The equivalences discussed should be reminiscent of set theory, highlighting the isomorphism between sets and functions.

- (77)
- $A = \{x \mid x \in A\}$
  - $\mathbf{dog} = \lambda x.\mathbf{dog}(x)$
  - $\{x \mid x \in A\} = \{y \mid y \in A\}$
  - $\lambda y.\mathbf{cat}(y) = \lambda x.\mathbf{cat}(x)$

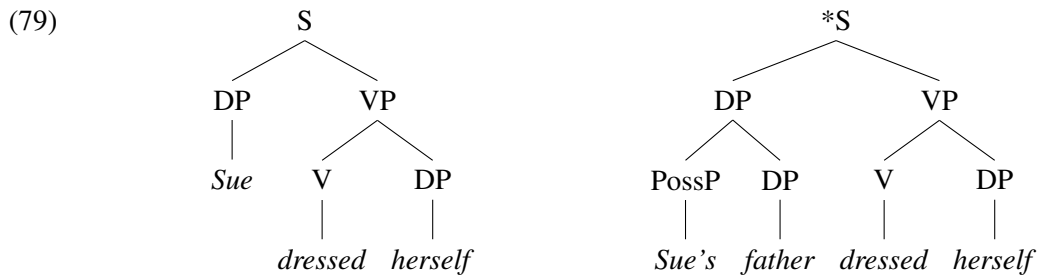
<sup>4</sup>so long as  $u$  is not free in  $\phi$

## 7.5.3 Case study 2: Reflexives

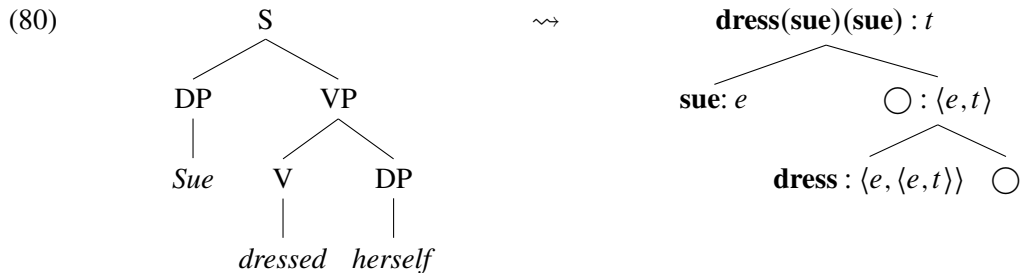
How are English sentences with *-self* pronouns in object position interpreted?

- (78) a. The child dressed herself.  
 b. John helped himself.  
 c. I saw myself (in the mirror).  
 d. Sue's father dressed \*herself/himself.

Conventional wisdom (from Lees and Klima 1963): a *-self* pronoun has to be 'co-referential' with a c-commanding DP. This means that the DP's sister must dominate the *-self* pronoun. If this structural relationship doesn't hold, reflexive-'binding' is impossible.



Using functional meanings, we can *derive* this analysis. We can provide a partial explanation of the informal 'binding principles' of, e.g., Chomsky 1981.

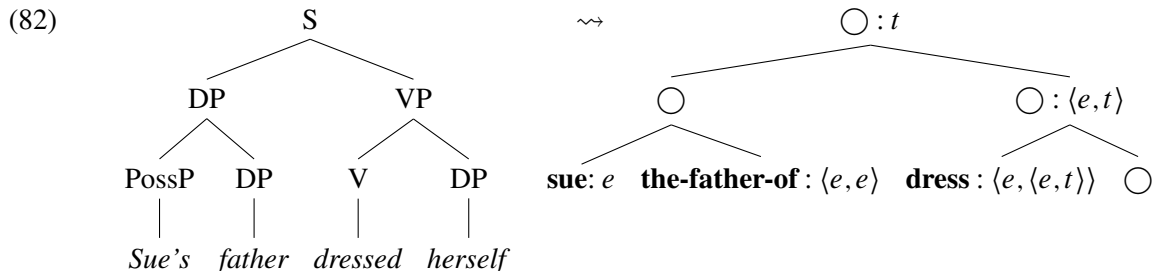


This analysis allows us to see why *Sue* can't antecede *herself* in (78-d). First a semantics for *father*:

- (81) *father*  $\rightsquigarrow$  **the.father.of** :  $\langle e, e \rangle$

Describe what kind of function **[[the-father-of]]** is.

Explain why the account in (80) explains the 'c-command' generalization.



## 7.6 Revisiting connectives

Our type space allows the type  $t$ , and therefore  $\langle t, t \rangle$ , and therefore  $\langle t, \langle t, t \rangle \rangle$ ,  $\langle \langle t, t \rangle, t \rangle$ , *ad infinitum*.

We have an obvious use for functions from truth values to truth values: connectives!

Earlier we proposed a type  $\langle \langle e, t \rangle, \langle e, t \rangle \rangle$  negation function. But what about a type- $\langle t, t \rangle$  version?

$$(83) \quad \llbracket \neg \rrbracket = \begin{bmatrix} \mathbf{T} & \mapsto \mathbf{F} \\ \mathbf{F} & \mapsto \mathbf{T} \end{bmatrix}$$

This might come in handy for sentential modifiers like:

- (84) a. *it's false that*  $\rightsquigarrow \neg : \langle t, t \rangle$   
 b. *it's not the case that*  $\rightsquigarrow \neg : \langle t, t \rangle$

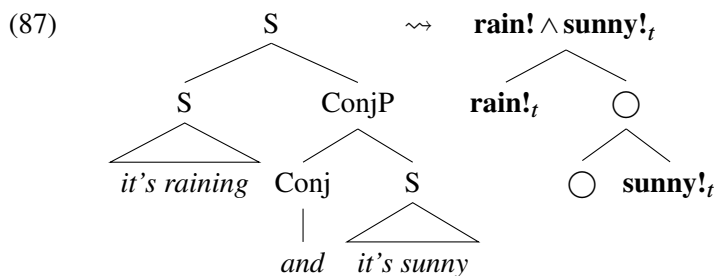
Or languages which have negation attaching at the S-level, like Māori.

- (85) Kāore [i-mau tētahi tuna kotahi i a Tamahae].  
 not PAST-catch INDEF eel one by PERS Tamahae  
 Tamahae didn't catch one eel.

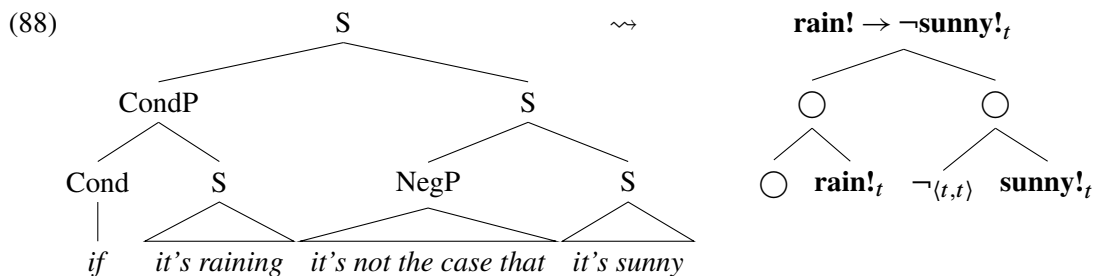
Chung and Ladusaw 2004:28

- (86) *kāore*  $\rightsquigarrow \neg : \langle t, t \rangle$

Let's propose meanings for the other connectives.



What about *or*?



So we have two negations to play with. Below they have  $\lambda$ s filled in,

- (89) a. *not*  $\rightsquigarrow \lambda p_t. \neg(p) : \langle t, t \rangle$   
 b. *not*  $\rightsquigarrow \lambda f_{\langle e, t \rangle}. \lambda x_e. \neg(f(x)) : \langle \langle e, t \rangle, \langle e, t \rangle \rangle$

As you saw in Homework 4, *and* isn't always at the sentence level.

- (90) a. No child skateboards and no child studies.  
b. No child skateboards and studies.
- (91) a. Every child skateboards and every child studies.  
b. Every child skateboards and studies.

We also need it for the ambiguity of the following:

- (92) Frowny is not tall and thin.
- (93) a. Where is the VP-level *and* attaching?  
b. What would it's type be (assuming that the conjoined VPs are type  $\langle e, t \rangle$ )?  
c. What's a ML translation for VP-level *and*?  
d. What about *or*?

## 7.7 Modifiers as functions

Let's revisit adjectival modifiers. In Handout 6, we conceptualized adjectives as a function from an input [N]-meaning to an output [Adj N]-meaning. What type would an adjective be?

This type coheres with our earlier conceptualization of adjectives as functions from sets to sets.

$$(94) \quad \llbracket \text{skillful} \rrbracket^M = \left[ \begin{array}{l} \left\{ \begin{array}{c} \text{☺}, \text{☹}, \text{☹} \end{array} \right\} \mapsto \left\{ \begin{array}{c} \text{☺}, \text{☹} \end{array} \right\} \\ \left\{ \begin{array}{c} \text{☹}, \text{☹} \end{array} \right\} \mapsto \left\{ \begin{array}{c} \text{☹}, \text{☹} \end{array} \right\} \\ \left\{ \begin{array}{c} \text{☺}, \text{☹} \end{array} \right\} \mapsto \left\{ \begin{array}{c} \text{☺} \end{array} \right\} \end{array} \right]$$

But **TY** doesn't use sets, so we have to convert any set to a function. What would this look like?

- (95)  $\llbracket \text{skillful} \rrbracket$  is a function mapping CFs (in  $D_{\langle e, t \rangle}$ ) to CFs (in  $D_{\langle e, t \rangle}$ ), such that for any  $d$  and any  $f$ , if  $\llbracket \text{skillful} \rrbracket(f)(d) = \mathbf{T}$ , then  $f(d) = \mathbf{T}$ .
- (96)  $\llbracket \text{alleged} \rrbracket$  is a function mapping CFs (in  $D_{\langle e, t \rangle}$ ) to CFs (in  $D_{\langle e, t \rangle}$ ), such that...
- (97)  $\llbracket \text{former} \rrbracket$  is a function mapping CFs (in  $D_{\langle e, t \rangle}$ ) to CFs (in  $D_{\langle e, t \rangle}$ ), such that for any  $d$  and any  $f$ , if  $\llbracket \text{alleged} \rrbracket(f)(d) = ?$ , then  $f(d) = ?$ .

(95) holds all subsecutive adjectives, including intersective adjectives like *Swedish*. But intersective adjectives can be given an even more refined semantics.

$$(98) \quad \text{Swedish} \rightsquigarrow \lambda g. \lambda x. \text{swedish}(x) \wedge g(x) : \langle \langle e, t \rangle, \langle e, t \rangle \rangle$$

Explain why this ML translation for *Swedish* ensures that (95) holds.

Can we do the same for *skillful*? Which of the following works better?

- (99) a.  $\text{skillful} \rightsquigarrow \text{skillful} : \langle \langle e, t \rangle, \langle e, t \rangle \rangle$   
b.  $\text{skillful} \rightsquigarrow \lambda g. \lambda x. \text{skillful}(x) \wedge g(x) : \langle \langle e, t \rangle, \langle e, t \rangle \rangle$

Remember the invalid inference from Handout 6:

- (100) Smiley is a skillful student.

Smiley is a violinist.  
 #Smiley is a skillful violinist.

## 7.8 Composing quantifiers

In Handout 5 we conceptualized quantificational determiners as relations between sets:

- (101) a.  $\llbracket \text{every} \rrbracket = \{ \langle A, B \rangle \mid A \subseteq B \}$   
 b.  $\llbracket \text{some} \rrbracket = \{ \langle A, B \rangle \mid A \cap B \neq \emptyset \}$

Of course, **TY** uses CFs instead of sets, so we'll have to use the same trick as we did for subjective adjectives. Let's convert (101-a) to be fully functional one step at a time.

First let's specify a function from sets to their supersets (informally, we'll think of (101-a) as a function from  $A$  to the set of possible  $B$ s).

- (102)  $\llbracket \text{every} \rrbracket =$  the function  $Det$  s.t. for any  $f \subseteq U$ ,  $Det(f)$  is the set of sets  $g$ , s.t.  $f \subseteq g$ .

(103)  $\llbracket \text{every} \rrbracket^M = \left[ \begin{array}{l} \{ \text{☺}, \text{☹} \} \mapsto \{ \{ \text{☺}, \text{☹} \} \} \\ \{ \text{☺} \} \mapsto \{ \{ \text{☺}, \text{☹} \}, \{ \text{☺} \} \} \\ \{ \text{☹} \} \mapsto \{ \{ \text{☺}, \text{☹} \}, \{ \text{☹} \} \} \\ \emptyset \mapsto \{ \{ \text{☺}, \text{☹} \}, \{ \text{☹} \}, \{ \text{☺} \}, \emptyset \} \end{array} \right]$

- (104) Assuming that  $\llbracket \text{saxophonist} \rrbracket = \{ \text{☺} \}$  and  $\llbracket \text{swimmer} \rrbracket = \{ \text{☺}, \text{☹} \}$ , what is  $\llbracket \text{every}(\text{saxophonist}) \rrbracket$ ?

What type do you think **every(saxophonist)** is?

The next step is reconfiguring the output of  $\llbracket \text{every} \rrbracket$  as a CF over sets. How would you describe the following CF in words?

(105)  $\llbracket \text{every}(\text{saxophonist}) \rrbracket^M = \left[ \begin{array}{l} \{ \text{☺}, \text{☹} \} \mapsto \mathbf{T} \\ \{ \text{☺} \} \mapsto \mathbf{T} \\ \{ \text{☹} \} \mapsto \mathbf{F} \\ \emptyset \mapsto \mathbf{F} \end{array} \right]$

- (106)  $\llbracket \text{every}(\text{saxophonist}) \rrbracket^M =$  the function  $Q$  such that

The final step is converting any sets of individuals to CFs. We've already seen how to do this.

- (107)  $\llbracket \text{saxophonist} \rrbracket =$  the function  $f$  such that for any  $d \in U$ ,  $f(x) = \mathbf{T}$  iff  $d$  is a saxophonist.

Now we can give a semantics for **every** in **TY**.

- (108) **every** is a function  $Det \in D_{\langle\langle e,t \rangle, \langle\langle e,t \rangle, t \rangle\rangle}$  such that for any  $f \in D_{\langle e,t \rangle}$ ,  $Det(f)$  is the function  $Q$ , such that for any  $g \in D_{\langle e,t \rangle}$ ,  $Q(g) = \mathbf{T}$  iff for any  $d \in D_e$ , if  $f(x) = \mathbf{T}$ , then  $g(x) = \mathbf{T}$ .

Obviously our previous denotation for **every** was much simpler and readable. Thankfully, we can use a notational shortcut to switch back and forth between CFs and sets.

- (109) If  $f \in D_{\langle e,t \rangle}$ , then  $f_* = \{x \mid f(x) = \mathbf{T}\}$

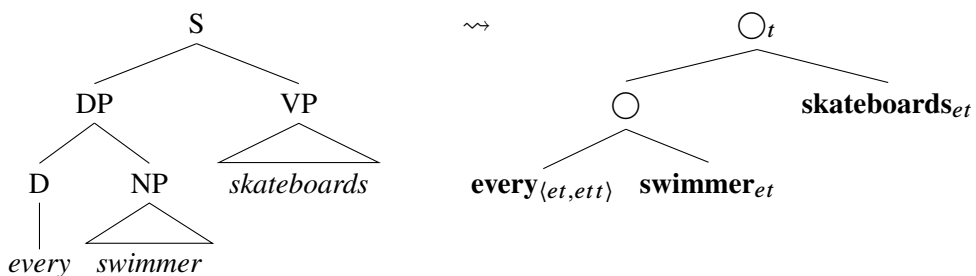
- (110)  $[[\text{swimmer}]]^M = \left[ \begin{array}{l} \text{☹} \mapsto \mathbf{T} \\ \text{☺} \mapsto \mathbf{T} \\ \text{☹} \mapsto \mathbf{F} \\ \text{☺} \mapsto \mathbf{F} \end{array} \right] \quad [[\text{swimmer}]]_*^M = \{ \text{☹}, \text{☺} \}$

A bit of terminology  $A$  is the characteristic function of the set  $A_*$ , and  $A_*$  is the characteristic set of the function  $A$ .

Using this notation, we can give functional denotations for our determiners:

- (111) a.  $[[\text{every}(P)(Q)]] = \mathbf{T}$  iff  $[[P]]_* \subseteq [[Q]]_*$   
 b.  $[[\text{some/any}(P)(Q)]] = \mathbf{T}$  iff  $[[P]]_* \cap [[Q]]_* \neq \emptyset$   
 c.  $[[\text{not every}(P)(Q)]] = \mathbf{T}$  iff  $[[P]]_* \not\subseteq [[Q]]_* \neq \emptyset$   
 d.  $[[\text{no}(P)(Q)]] = \mathbf{T}$  iff  $[[P]]_* \cap [[Q]]_* = \emptyset$   
 e.  $[[\text{most}(P)(Q)]] = \mathbf{T}$  iff  $|[[P]]_* \cap [[Q]]_*| > |[[P]]_* - [[Q]]_*|$   
 f.  $[[\text{at least three}(P)(Q)]] = \mathbf{T}$  iff  $|[[P]]_* \cap [[Q]]_*| \geq 3$

Let's make sure we understand (also note the common type-abbreviations):

- (112) 

Give a set of plausible and consistent denotations for each node:

- (113) a. VP  
 b. NP  
 c. D, as in (111-a)  
 d. DP  
 e. S

## 7.9 Possible paper topics

- The semantics of *-self* pronouns, reflexives of other varieties, and so on, is a goldmine for syntax-semantic research. The theory in this handout gets pretty far, but needs extension to reflexives in other positions, like in PPs and subjects of infinitives. The theory in this handout comes from Barker 2016; Barker and Shan 2014; Winter 2016. Some good resources for the syntactic side, Kiparsky 2002 and Charnavel and Sportiche 2016. There's also the issue of reciprocals like *each other* (see Dalrymple et al. 1998), and unbound reflexives (see Charnavel 2018).
- Another big issue touched upon here, the syntax of negation. See Kim and Sag 2002; Sag et al. 2019; Zeijlstra 2004, and many others. Do all languages have this dichotomy between  $\langle t, t \rangle$ -negation and sub-sentential negation? What about other connectives?

## 7.10 Further reading

- Much of the material here comes from Winter 2016:§3–4, and Potts 2007:§4–6.
- For more detail about all these issues, I recommend Carpenter 1997:§1 and Cann, Kempson, and Gregoromichelaki 2009:§3.
- An alternative view on the issues raised here is found in Heim and Kratzer 1998. Their semantic theory is very similar in appearance to the one proposed here, but they collapse notions of a metalanguage and model, and (presumably for this reason) their interpretation function  $\llbracket \ ]$  takes natural language as its input and returns  $\lambda$ -expressions as its output.

## Bibliography

- Barker, Chris. 2016. "Evaluation order, crossover, and reconstruction". Ms., NYU.
- Barker, Chris, and Chung-Chieh Shan. 2014. *Continuations and Natural Language*. Oxford: Oxford University Press.
- Bentham, Johan van, et al. 2016. "Logic in Action". Available at [www.logicinaction.org](http://www.logicinaction.org).
- Cann, Ronnie, Ruth Kempson, and Eleni Gregoromichelaki. 2009. *Semantics: An Introduction to Meaning in Language*. Cambridge: Cambridge University Press.
- Carpenter, Bob. 1997. *Type-Logical Semantics*. Cambridge, MA: MIT Press.
- Charnavel, Isabelle. 2018. "Logophoricity and locality: A view from French anaphors". Ms., Harvard University.
- Charnavel, Isabelle, and Dominique Sportiche. 2016. "Anaphor binding: What French inanimate anaphors show". *Linguistic Inquiry* 47 (1): 35–87.
- Chung, Sandra, and William A. Ladusaw. 2004. *Restriction and Saturation*. Cambridge, MA: MIT Press.
- Dalrymple, Mary, et al. 1998. "Reciprocal expressions and the concept of reciprocity". *Linguistics and Philosophy* 21 (2): 159–210.
- Gallin, Daniel. 1975. *Intensional and Higher Order Modal Logic*. Amsterdam: North-Holland.
- Heim, Irene, and Angelika Kratzer. 1998. *Semantics in Generative Grammar*. Oxford: Blackwell.
- Kim, Jong-Bok, and Ivan A. Sag. 2002. "Negation without head-movement". *Natural Language and Linguistic Theory* 20:339–412.
- Kiparsky, Paul. 2002. "Disjoint reference and the typology of pronouns". In *More than Words*, edited by I. Kaufmann and B. Stiebels, 179–226. Berlin: Akademie Verlag.
- Potts, Christopher. 2007. "Logic for Linguists". Available at [www.christopherpotts.net/ling/teaching/lsa108P](http://www.christopherpotts.net/ling/teaching/lsa108P).



Sag, Ivan A., et al. 2019. "Lessons from the English auxiliary system". To appear in *Journal of Linguistics* 56.

Winter, Yoad. 2016. *Elements of Formal Semantics*. Edinburgh: Edinburgh University Press.

Zeijlstra, Hedde. 2004. "Sentential negation and negative concord". PhD thesis, University of Amsterdam.